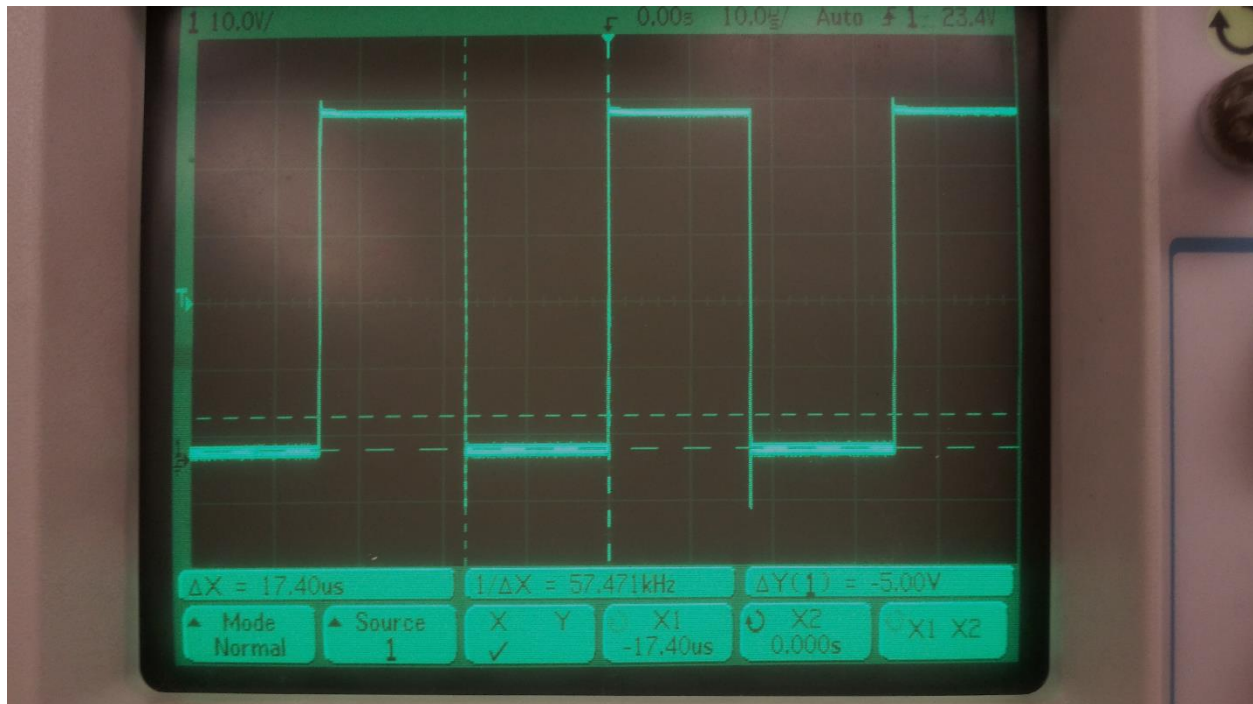


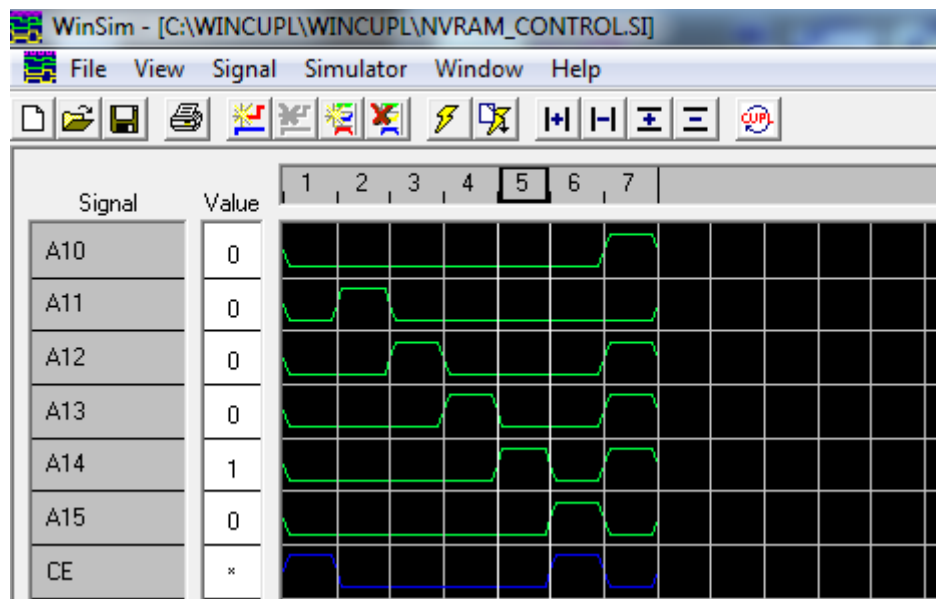
## Required Elements

### Determining Fastest Baudrate between Host Computer and 8051:



To test how fast I can send data to my host machine, I wrote a send 'U' program that sends data at a baud rate of 57600. You can see in the signal trace above with a frequency of 57.471 kHz. This is the baud rate of the data being sent. You can also see a sequence of 0's and 1's since the letter U is being sent continuously.

### NVRAM Data Memory Setup



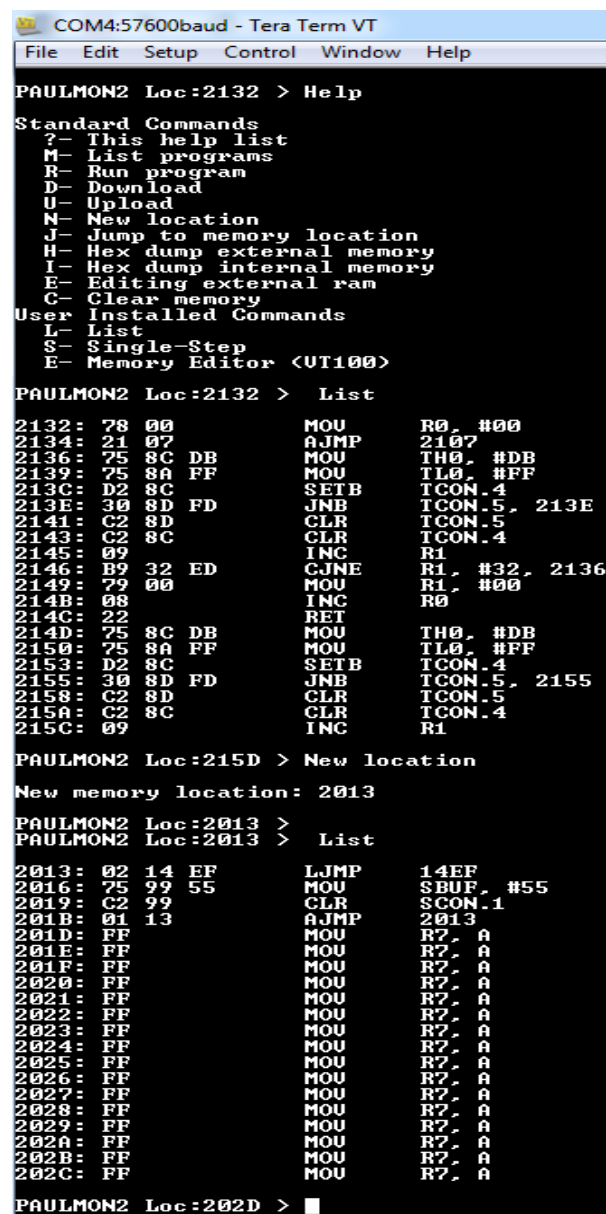
If you take a look at the my spld code you can see the logic I used to ensure data writes only happen between 0x400 to 0x7FFF. The image above is a simulation of that logic. The logic basically looks at addresses A10 – A15. The logic is as follows:

```
FIELD ADDR = [A15..A10];
```

```
CE = !(ADDR:#) # A15;
```

CE is the control signal that turns the NVRAM on or off. If all the address bits from A15 to A10 are 0, then the memory will be less than 0x400. Thus, CE must go high (active low signal). In addition, if A15 ever goes high, then CE must go high since the address is 0x8000 or larger. You can see the desired outcome in the simulation above. As long as any address bit is 1, then the address will be in the correct address range, unless A15 goes high.

### Using Paulmon2:



```
COM4:57600baud - Tera Term VT
File Edit Setup Control Window Help

PAULMON2 Loc:2132 > Help
Standard Commands
?- This help list
M- List programs
R- Run program
D- Download
U- Upload
N- New location
J- Jump to memory location
H- Hex dump external memory
I- Hex dump internal memory
E- Editing external ram
C- Clear memory
User Installed Commands
L- List
S- Single-Step
E- Memory Editor <UT100>

PAULMON2 Loc:2132 > List
2132: 78 00      MOU    R0, #00
2134: 21 07      AJMP   2107
2136: 75 8C DB    MOU    TH0, #DB
2139: 75 8A FF    MOU    TL0, #FF
213C: D2 8C      SETB   TCON.4
213E: 30 8D FD     JNB    TCON.5, 213E
2141: C2 8D      CLR     TCON.5
2143: C2 8C      CLR     TCON.4
2145: 09         INC     R1
2146: B9 32 ED     CJNE   R1, #32, 2136
2149: 79 00      MOU    R1, #00
214B: 08         INC     R0
214C: 22         RET
214D: 75 8C DB    MOU    TH0, #DB
2150: 75 8A FF    MOU    TL0, #FF
2153: D2 8C      SETB   TCON.4
2155: 30 8D FD     JNB    TCON.5, 2155
2158: C2 8D      CLR     TCON.5
215A: C2 8C      CLR     TCON.4
215C: 09         INC     R1

PAULMON2 Loc:215D > New location
New memory location: 2013
PAULMON2 Loc:2013 >
PAULMON2 Loc:2013 > List
2013: 02 14 EF    LJMP   14EF
2016: 75 99 55    MOU    SBUF, #55
2019: C2 99      CLR     SCON.1
201B: 01 13      AJMP   2013
201D: FF         MOU    R7, A
201E: FF         MOU    R7, A
201F: FF         MOU    R7, A
2020: FF         MOU    R7, A
2021: FF         MOU    R7, A
2022: FF         MOU    R7, A
2023: FF         MOU    R7, A
2024: FF         MOU    R7, A
2025: FF         MOU    R7, A
2026: FF         MOU    R7, A
2027: FF         MOU    R7, A
2028: FF         MOU    R7, A
2029: FF         MOU    R7, A
202A: FF         MOU    R7, A
202B: FF         MOU    R7, A
202C: FF         MOU    R7, A
PAULMON2 Loc:202D > █
```

The image above is just showing basic commands of Paulmon2. It has a help menu that says which commands are available. As you can, I can view the flash memory and list the assembly instructions at each memory location.

```

COM4:57600baud - Tera Term VT
File Edit Setup Control Window Help
DATA      8051 External Memory Editor, Paul Stoffregen, 1996
ADDR: +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F ASCII EQUIVILANT
7F00: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F10: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F20: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F30: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F40: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F50: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F60: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F70: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F80: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7F90: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7FA0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7FB0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7FC0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7FD0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7FE0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
7FF0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
^A=ASCII ^X=Hex ^F=Fill ^G=Goto ^C=Code ^D=Data ^L=Redraw ^Q=Quit

```

```

DATA      8051 External Memory Editor, Paul Stoffregen, 1996
ADDR: +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F ASCII EQUIVILANT
8000: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8010: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8020: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8030: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8040: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8060: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8070: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8080: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8090: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
80A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
80B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
80C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
80D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
80E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
80F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
^A=ASCII ^X=Hex ^F=Fill ^G=Goto ^C=Code ^D=Data ^L=Redraw ^Q=Quit

```

```

DATA      8051 External Memory Editor, Paul Stoffregen, 1996
ADDR: +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F ASCII EQUIVILANT
0000: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0010: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0020: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0030: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0040: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0050: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0060: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0070: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0080: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
0090: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
00A0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
00B0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
00C0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
00D0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
00E0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
00F0: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUUUUU
^A=ASCII ^X=Hex ^F=Fill ^G=Goto ^C=Code ^D=Data ^L=Redraw ^Q=Quit

```

The image above just shows that I am able to use the external memory editor in Paulmon2. I am able to fill 55's in the memory 0x0000 to 0x7FFF. I was not able to write memory past 0x7FFF, which was expected.

### Paulmon2 Challenge:

Paulmon2 has a single step function. It does not work out of the box because the Paulmon2 code makes an assumption that the program memory is in external memory. This is not a major issue because Paulmon2 only makes a single change to the program memory. That change is placing an `ljmp` to "step" in the `int1` interrupt. Step is basically an ISR that prints out program information after each step in the program. Paulmon2 is leveraging the fact that an instruction is always run after an interrupt returns. So, to create a single step feature, they use an interrupt to print out data and program information and waits on the user to press the spacebar. Once they do, it leaves the returns from the routine, an instruction is run, and prints out the data and program information.

To allow this functionality to work with Paulmon2 and the program to sit in flash memory, I simply write a single line of code in the program that is debugged. I write the address of step in the `int 1` interrupt, so even when Paulmon fails to write the address of step in flash memory (actually address of step is just written in external memory) the `int1` knows where to jump to.

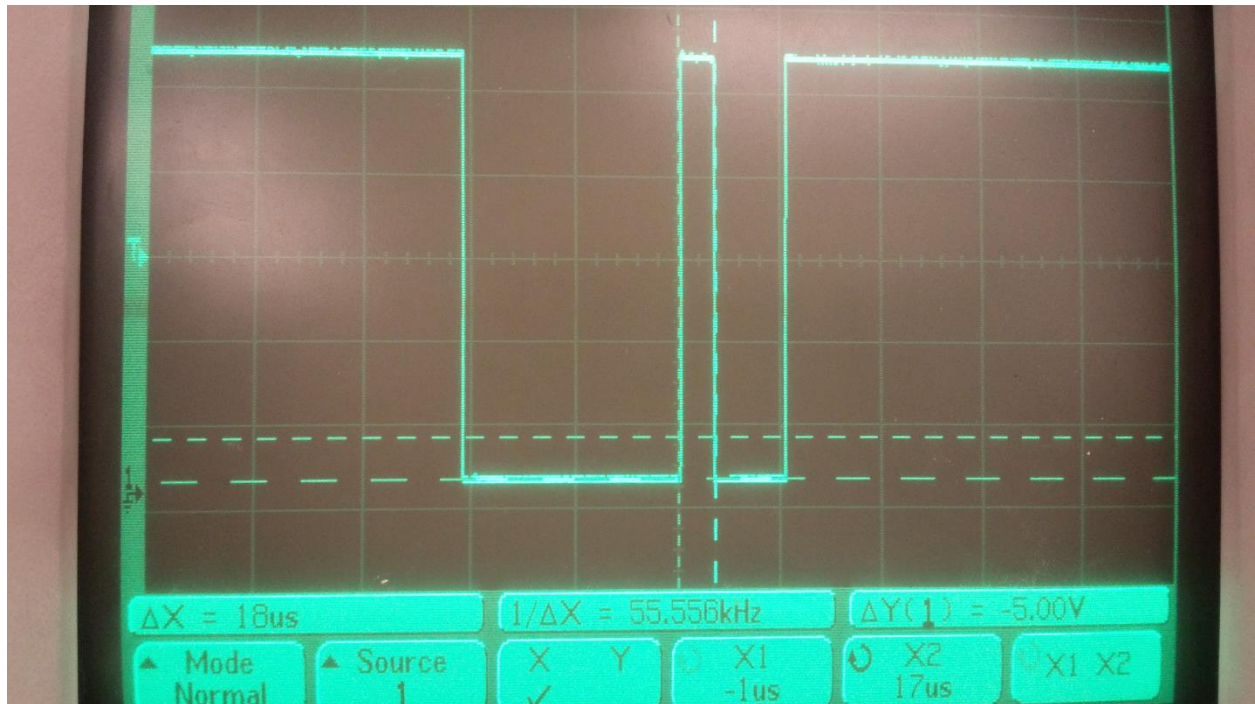
The code I used Paulmon2 to debug is called "SOSDEBUG.ASM". The images below shows the debugger working correctly and how to run the program.

```
Jump to memory location <0000>, or <ESC> to exit: 2100
Now running in single step mode: <RET>= step, ?= Help
```

ACC	B	C	DPTR	R0	R1	R2	R3	R4	R5	R6	R7	SP	Addr	Instruction
00	00	0	2100	00:00:00:00:00:00:00:00	0A	14DD:	JMP	EA+DPTR						
00	00	0	2100	00:00:00:00:00:00:00:00	0A	2100:	MOU	TMOD, #01						
00	00	0	2100	00:00:00:00:00:00:00:00	0A	2103:	MOU	R0, #00						
00	00	0	2100	00:00:00:00:00:00:00:00	0A	2105:	MOU	R1, #00						
00	00	0	2100	00:00:00:00:00:00:00:00	0A	2107:	SETB	P1.2						
00	00	0	2100	00:00:00:00:00:00:00:00	0C	2109:	LCALL	2136						
00	00	0	2100	00:00:00:00:00:00:00:00	0C	2136:	MOU	TH0, #DB						
00	00	0	2100	00:00:00:00:00:00:00:00	0C	2139:	MOU	TL0, #FF						
00	00	0	2100	00:00:00:00:00:00:00:00	0C	213C:	SETB	TCON.4						
00	00	0	2100	00:00:00:00:00:00:00:00	0C	213E:	JNB	TCON.5, 213E						
00	00	0	2100	00:00:00:00:00:00:00:00	0C	2141:	CLR	TCON.5						
00	00	0	2100	00:00:00:00:00:00:00:00	0C	2143:	CLR	TCON.4						
00	00	0	2100	00:01:00:00:00:00:00:00	0C	2145:	INC	R1						
00	00	1	2100	00:01:00:00:00:00:00:00	0C	2146:	CJNE	R1, #32, 2136						
00	00	1	2100	00:01:00:00:00:00:00:00	0C	2136:	MOU	TH0, #DB						
00	00	1	2100	00:01:00:00:00:00:00:00	0C	2139:	MOU	TL0, #FF						
00	00	1	2100	00:01:00:00:00:00:00:00	0C	213C:	SETB	TCON.4						
00	00	1	2100	00:01:00:00:00:00:00:00	0C	213E:	JNB	TCON.5, 213E						
00	00	1	2100	00:01:00:00:00:00:00:00	0C	2141:	CLR	TCON.5						
00	00	1	2100	00:01:00:00:00:00:00:00	0C	2143:	CLR	TCON.4						
00	00	1	2100	00:02:00:00:00:00:00:00	0C	2145:	INC	R1						
00	00	1	2100	00:02:00:00:00:00:00:00	0C	2146:	CJNE	R1, #32, 2136						
00	00	1	2100	00:02:00:00:00:00:00:00	0C	2136:	MOU	TH0, #DB						
00	00	1	2100	00:02:00:00:00:00:00:00	0C	2139:	MOU	TL0, #FF						

```
PAULMON2 Loc:2100 > Jump to memory location
Jump to memory location <2100>, or ESC to quit: 2100
running program:
```

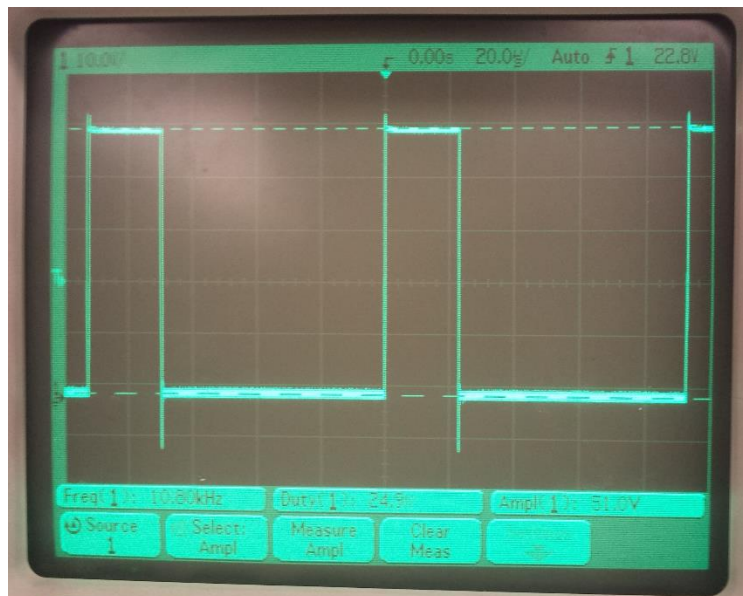
### Max Baud Rate Paulmon2 can run on:



The oscilloscope trace above shows roughly a baud rate of 57600. This is the highest rate I could reliably run Paulmon2 at. I tried the 115200 baud rate, however, Paulmon2 did not respond.

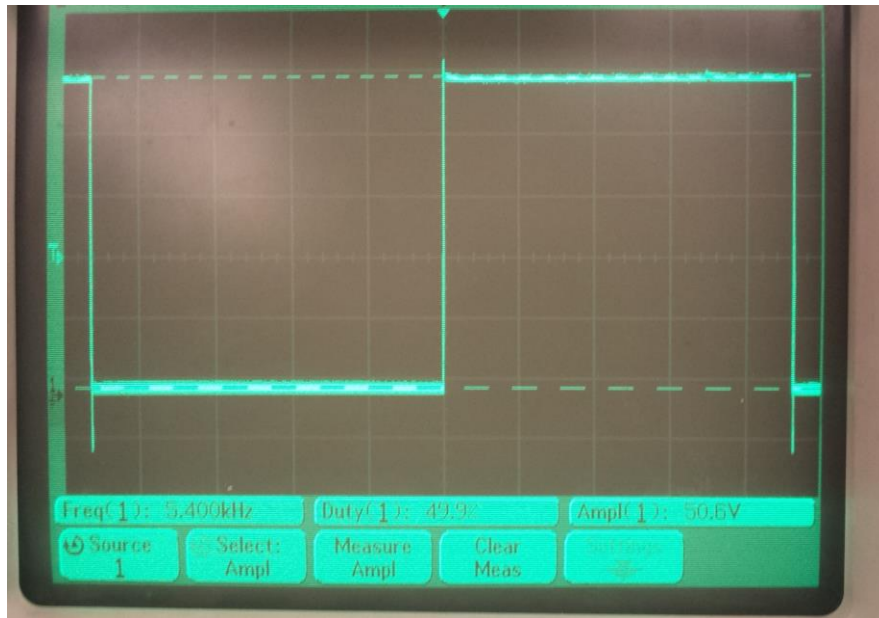
### Supplemental Elements:

#### PWM



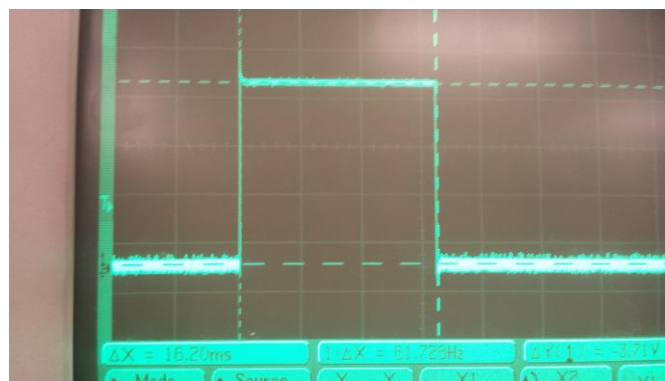
The oscilloscope trace above shows a PWM signal with a 25% duty cycle. By leveraging the PCA, I set Module 0 as a PWM signal generator. I configured the PCA to clock on  $\text{Clk}_{\text{periph}}/2$ . The PCA contains two registers that contain the count, CL and CH. More specifically and in the case of PWM, the CL count register is compared to a pre-set register CCAP0L. While CL is less than this value, the output of PWM will be low. Once CL is equal or greater than CCAP0L, then it will set the signal to high. Note that CCAP0L is auto-reloaded once CL overflows. CCAP0L is loaded with the value in CCAP0H. CCAP0H is ultimately what allows us to control the duty cycle.

### High-speed Output



The oscilloscope trace above shows the output of Module 1 of my PCA. It was utilized as a high speed output. The high speed output is very similar to PWM, however, it does a 16-bit compare and its main functionality is toggling. When the 16-bit value represented with registers CCAP0H and CCAP0L match CH and CL, then it will toggle its output pin. Even when the CH and CL registers roll over, the high-speed output module will continue to wait until there is a match.

### Floating Point Challenge:



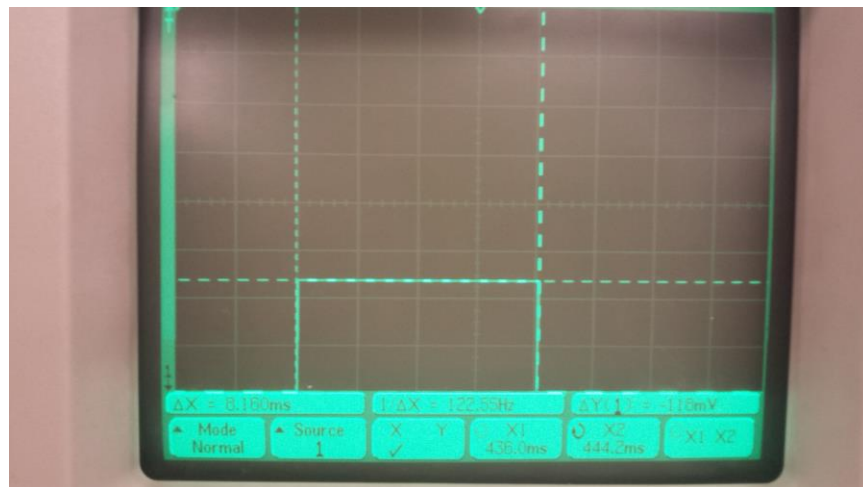


The oscilloscope trace above shows the time it took the 8051 to complete the following calculation in X1 mode. You can see that it takes 16.20 ms . This measurement was done by toggling an output pin (P1.2) before and after the match calculation.

$$\text{float } a = 1.234234234$$

$$\text{float } b = 7.225423556$$

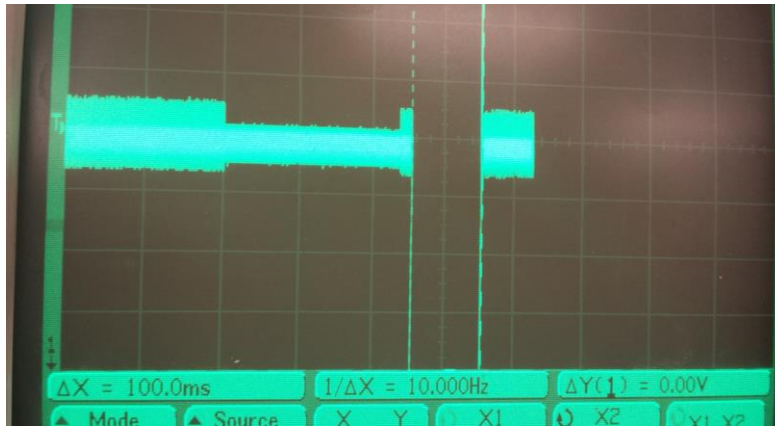
$$\text{Answer} = \sqrt{(((a + b)/b) * 6.2) * \left(\left(\frac{a + b}{b}\right) * 6.2\right)^3 * 10.6}$$



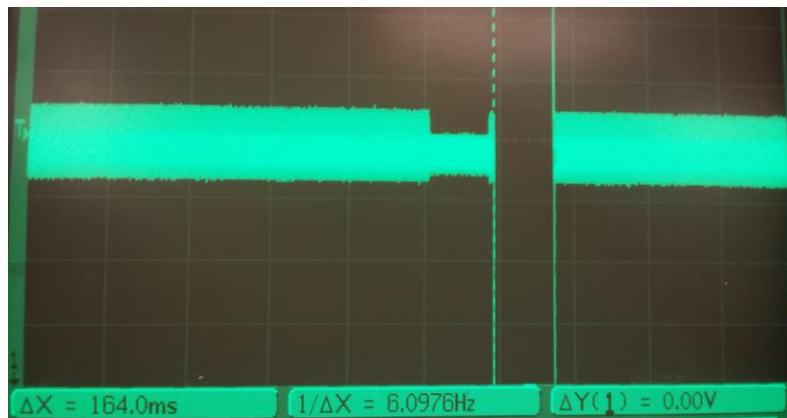
The oscilloscope trace above shows the time it took the 8051 to complete the same calculation above in X2 mode. You can see that it takes 8.16 ms, which is roughly half the time it takes in X1 mode. This is expected since the number of oscillations in a clock cycle is now 6, instead of 12 in X1 mode. This measurement was done by toggling an output pin (P1.2) before and after the match calculation.

### Serial Challenge

In order to show that there is a performance increase in using FIFOs and serial port interrupt instead of polling on the TI and RI flags, both techniques were utilized to transmit data. In theory, the FIFO and serial interrupt should have better performance, since the program can just push data onto a FIFO and continue executing the rest of the instructions instead of waiting for each character to be successfully sent. A program was written to transmit 10 lines to terminal then compute the floating point calculation from the math challenge. This program also toggles a pin before and after the transmits and calculation, allowing the time to be measured. This program was run once using the FIFOs and serial interrupts and again with polling on TI and RI. The first oscilloscope trace shows the time FIFO implementation and it takes 100ms. The second oscilloscope shows the time for the polling implementation and it takes 164 ms. As expected, the FIFO implementation performed better.



**FIFO Implementation**



**Polling Implementation**

**Lab 3 Questions:**

1. Windows 7
2. SDCC 2.6
3. CodeBlocks 12.11
4. No
5. No